25th Annual Software Engineering Workshop
Goddard Space Flight Center, Maryland, 28-30 November 2000

Using Models to Test Process Assumptions within the SEL Recommended Software Development Approach[1]

Paolo Donzelli and Giuseppe Iazeolla
Laboratory for Computer Science
and CERTIA Research Center
University of Rome "TorVergata"
Roma, Italy
donzelli,iazeolla@info.uniroma2.it

Keywords:
Software Process Improvement, Process Modeling, Hybrid Simulation

1. Introduction

Software Process Validation testing is the activity that tests the capability of a given software process to satisfy user needs.

Validation of process assumptions is one form of validation testing. It consists of ensuring that assumptions about the effects of introducing variants to the current paradigm of software development are correct. For example, a project manager may want to be sure, before adopting an extra review activity, that this will not carry any risk of schedule overrun; a testing manager may want to verify to have enough staff available before committing himself towards higher defect reduction performances; a process modeler may want to estimate the amount of process overhead that could be generated by introducing higher concurrency levels among process activities.

Software companies face the problem of validating process assumptions whenever they are unsatisfied of current productivity or quality levels, and are willing to experiment new development policies. Different assumptions on many process parameters can in fact strongly affect the final process quality. Such parameters range from the structure adopted for the particular process instantiation, according to the chosen paradigm (number and type of phases and activities, nature of the exchanged artifacts, etc.), to the allocation of defect detection resources (personnel, tools, and methods) throughout the lifecycle, to the priority policies assumed for different tasks (defect correction, requirements change, etc.).

In many cases, however, such validation has to be done without affecting the actual environment. To this purpose, we argue that the complexity of the software processes, their dynamic behavior, and the need to quantitatively and qualitatively evaluate alternative assumptions (or sets of assumptions) call the adoption of process models. The use of process models is in fact an effective means to test and validate new assumptions on process parameters [4,10].

To be effective, however, process models must combine the ability to sustain the complexity of

[1] This work is one of the results of the joint co-operation between the University of Roma "Tor Vergata", the Enterprise-University Consortium CERTIA, and the Software Engineering Laboratory of the University of Maryland.

the modeling problem with the so-called dynamic estimation capability, that is the capability of representing the dynamics of the simulated process. In other words, the ability of dealing with the inter-twinned effects of various internal and external process parameters, of reconstructing process trajectories over time under different what-if conditions, and of estimating process's outcomes in a perturbed environment [3, 5, 8].

In such a perspective, the thrust of this paper is twofold:

- Introduce an approach capable of leading to process models with the required dynamic estimation capability, suitable to act as process testing environments to allow researchers, process modelers, and project managers to view the implications of their assumptions;
- Describe a model of the software development process recommended by the Software Engineering Laboratory (SEL), as an application of the proposed modeling approach.

2. The Modelling Approach

To address process modeling issues, the paper proposes the combination of three traditional modeling methods (analytical, continuous and discrete-event) into a unique *hybrid two-level modeling approach* [6]. At the higher abstraction level, the process is modeled by a discrete-event queuing network, which represents the component activities (i.e. service stations), their interactions, and the exchanged artifacts. At the lower abstraction level, instead, the analytical and continuous methods are used, to describe the implementation details of the introduced activities.

Indeed, the software process shows both discrete system aspects (start/end of an activity, reception/release of an artifact by an activity) and continuous system ones (resources consumption by an activity, percentage of developed product), and thus the proposed modeling approach provides a way to hierarchically take into account such different aspects. Furthermore, its practical application shows that it provides a powerful method to accurately model and analyze software processes, helping in bridging the gap between modelers and process experts. The software process, in fact, is described in terms of highly representative graphical models (queuing networks), providing a better visibility of the modeled process, of its operational environment and managerial policies, whereas the component activities are described in terms of well known analytical and continuous models, easily and quickly updateable. We argue that the produced models are highly flexible, being easily adaptable to the characteristics and the maturity level of the production environment, and updateable to follow its evolution (as advocated by such software process improvement methods as the Capability Maturity Model or the Quality Improvement Paradigm [1]).

3. A Model of the SEL Recommended Software Development Approach

The proposed hybrid modeling approach is applied to *model the software development process recommended by SEL* [11], focusing on process quality attributes such as effort, delivery time, productivity, rework percentage, product defect density and some sub-attributes thereof (final product size, process staffing profile, staffing profile over single activities, defects pattern, etc.).

According to SEL, the software process (illustrated in Figure 1) consists of a series of sequential phases, and the software product is the conclusive artifact of a series of intermediate artifacts. In particular, we defined the following main artifacts: *requirements*, *specification*, *high-level design*, *low-level design*, *code*, *system-tested code* and *acceptance-tested code*.

Although phases are sequential, their respective activities can run concurrently, given the simultaneous execution of work activities (that generate the artifacts mentioned above) and rework activities (necessary to fix defects or introduce requirement modifications). Thus, we introduced artifacts generated and dealt with by activities aiming at fixing defects, i.e. *defects* *reports* and *corrections reports* (e.g. *low-level design defects reports*, *code correction reports*, etc.), and artifacts generated by activities that introduce modifications due to requirements instability, i.e. *changes* and *increments* (e.g. *specification changes, high-level design increments*, etc.).

The resulting modeled process thus consists of partly sequential and partly concurrent activities: some of which are development activities, i.e. *Specification* (SP), *High Level Design* (HLD), *Low Level Design* (LLD), and *Implementation* (IMP), and some others are testing activities, i.e. *System Test* (ST), and *Acceptance Test* (AT).

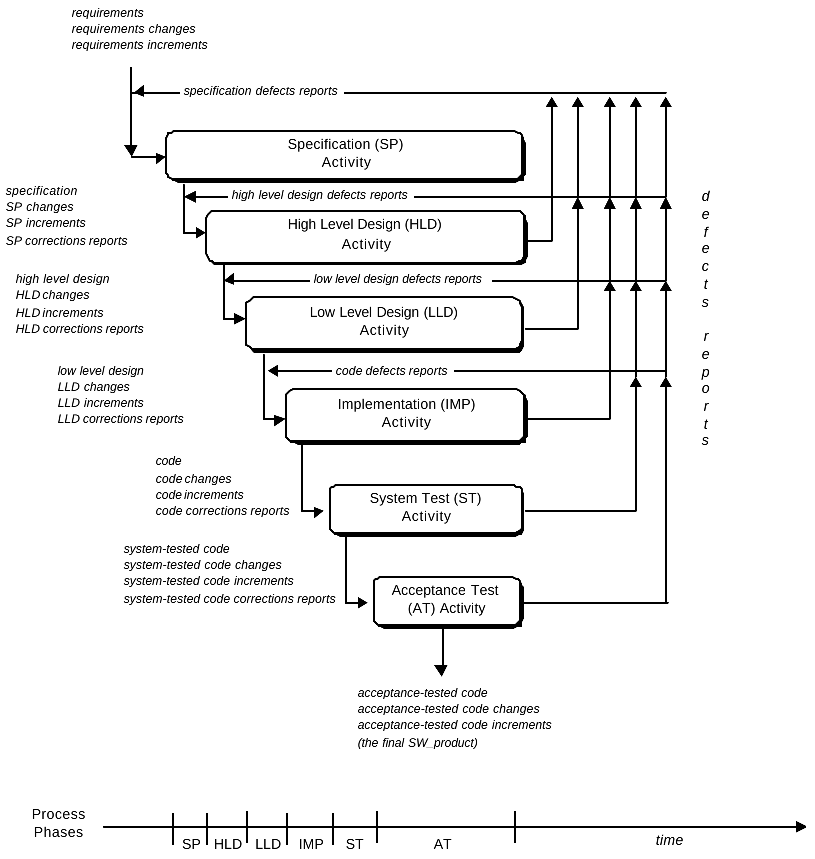


Figure 1 – the modelled software process

Applying our hybrid approach, the software process is translated into a two-level model, consisting of a higher and a lower abstraction level.

<u>At the higher abstraction level</u>, a queuing model is produced that is a direct replica of the software process, with service stations used to represent activities, and circulating customers used to represent artifacts that move from one activity to another. In particular, each activity is described by a set of service stations to represent the component sub-activities, e.g. development of the main artifact, defect correction, review, etc.. Two different types of queueing networks have been used to model the development activities (SP, HLD, LLD, IMP) and the testing activities (ST, AT), respectively.

As example of the development activities, Figure 2 illustrates the queueing network used to model the HLD activity. The main service stations of the HLD activity are the "work station", the "external rework station", the "internal rework station" and the "review station". The “work station” simulates the development of the *high-level design* artifact on the basis of the received *specification* artifact. Depending on the received *SP changes*, and *SP increments*, the “external rework station” simulates the modification of the already released *high-level design*, and yields the corresponding output artifacts (*HLD changes* and *HLD increments*). Similarly, on the basis of the received *SP corrections reports* and *HLD defects reports*, the “internal rework station” simulates the correction of the released *high-level design*, and yields the corresponding *HLD corrections reports*. Finally, the “review station” simulates the review performed on the *high-level design*, on the *HLD changes*, and *HLD increments*. No review is performed on the *HLD correction reports*, assumed with no defects. In other words, it is assumed that no further defects (bad fixes) are injected during the correction activities performed by the “internal rework station”.

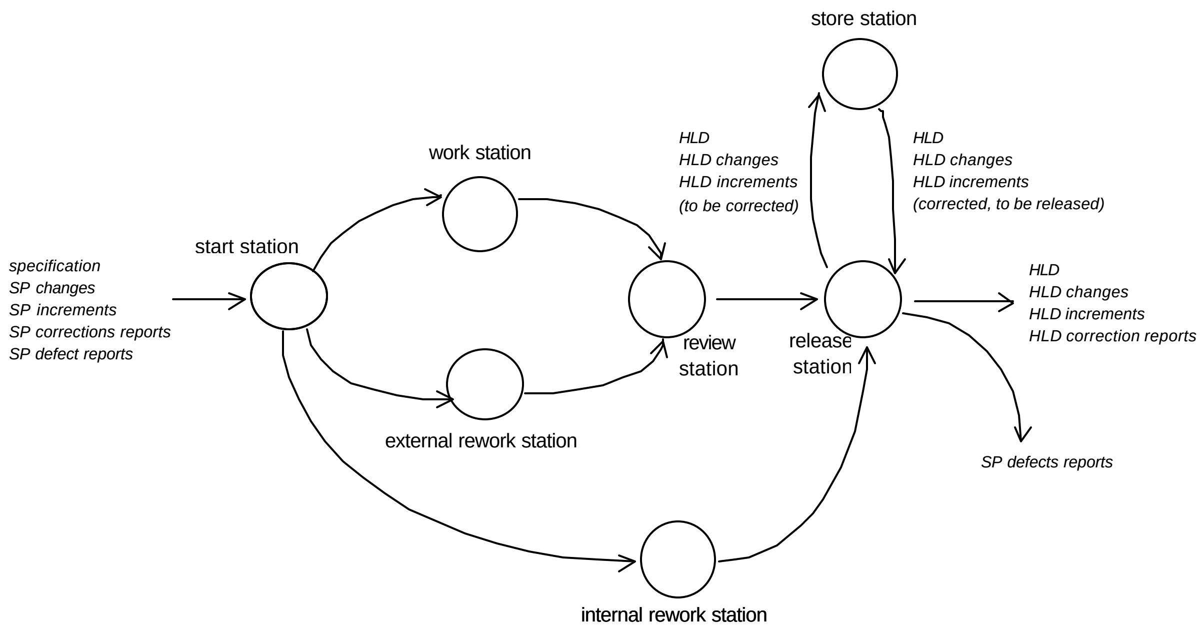


Figure 2 - Higher abstraction level of the HLD activity (a development activity)

As example of the testing activities, Figure 3 illustrates the queueing network used to model the AT activity. The main service stations are the "work testing station", and the "external rework testing station". The “work testing station” simulates the acceptance testing of the *system-tested code*, whereas the “external rework testing station” simulates the acceptance testing of the *system-tested code changes* and *increments*. Again, the *system-tested corrections reports* are assumed without defects.

In both the activities (figures 2 and 3), the “start”, the “release” and the “store” stations are assumed to be zero service-time stations, performing co-ordination activities only. The “start station” takes care of routing the input artifact to the appropriate service station, while the “release station” and the “store station” take care of the release of the artifacts.

At the lower abstraction level, the behavior of each service station introduced at the higher level is modeled by either an analytical average-type function, or by a continuous type time-varying function, or by a combination thereof. Such functions, mainly derived by the SEL models [7, 12, 13, 14], are used to express the amount of resources, or time, or effort that various service stations use to simulate the corresponding activities or sub-activities.

Figure 4 shows the implementation details of the HLD “work station”, the main of the service stations depicted in Figure 2, and of its corresponding input and output artifacts.

The station simulates the development of the *high-level design* artifact, starting from the *specification* artifact. The *specification* and *high-level design* artifacts are described by a set of four attributes: name, size, development effort and defectiveness, as in the squared frames on the top of Figure 4

Name and size are of immediate evidence. The defectiveness attribute is described by an array whose j-th element is the amount of defects injected into the artifact by the j-th development activity (j = SP, HLD, LLD, IMP). For example, for the case in Figure 4, the defectiveness of the *specification* artifact is given by D1= [D1(SP),0,0,0]. The development effort attribute (W1 for the *specification* and W1+W for the *high-level design*) is the effort that has been spent to develop the artifact itself since the beginning of the process. Thus, it encompasses also the effort spent to develop all the artifacts from which it has been derived.

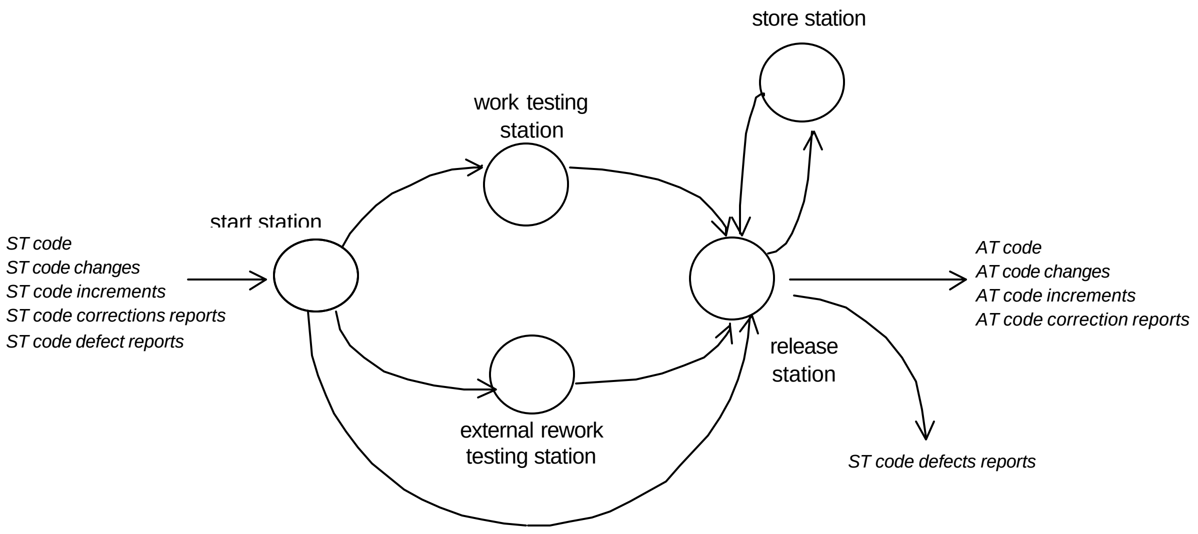


Figure 3 - Higher abstraction level of the AT activity (a testing activity)

The values of the attributes in the *high-level design* frame, together with the amount of time, T, (or the "work station" service time), and the required personnel over time, E(t), are derived by application of analytical average-type functions, and continuous type time-varying functions, as illustrated by the blocks in Figure 4. Such quantities may have random deviations, and are therefore simulated according to gaussian-like probability distributions.

More in detail, the average size of the *high-level design* artifact is first derived from the size of the *specification* artifact by use of COCOMO [2]-like size estimators.

$$average\_HLD\_size = a_1 SP\_size^{b_1} + c_1$$

The corresponding random *high-level design* size (*HLD_size*) is then obtained by use of the gaussian-like pseudo-random generator. This value is then given to the COCOMO-like time estimator block, to obtain the random release time (T), and to the COCOMO-like effort estimator, to obtain the random total development effort (Wtot).

$$T = a_2 HLD\_size^{b_2} + c_2$$

$$W_{tot} = a_3 HLD\_size^{b3} + c_3$$

On the basis of such T and Wtot, the effort density along time to produce the *high-level design*, E(t), is finally obtained using the Rayleigh function [9].

$$\mathrm{E(t)} = \mathrm{W_{tot}} \frac{\mathrm{t}}{\mathrm{T}^2} e^{-\frac{t^2}{2T^2}}$$

Unlimited staff availability is assumed. In other words, it is assumed that the staff pool in Figure 3 can always supply the personnel necessary to fit the E(t) curve demand for personnel. However more realistic assumptions on finite staff pools can be easily adopted.

According to Putnam's assumption [9], the *high-level design* artifact is released when E(t) reaches its peak. This means that the effort (W) simulated by the "work station" is a fixed fraction of Wtot, as shown by the shaded area in Figure 4.

This value (W) is then added to the development effort (W1) of the *specification* artifact to obtain the corresponding *high-level design* development effort (W1+W).

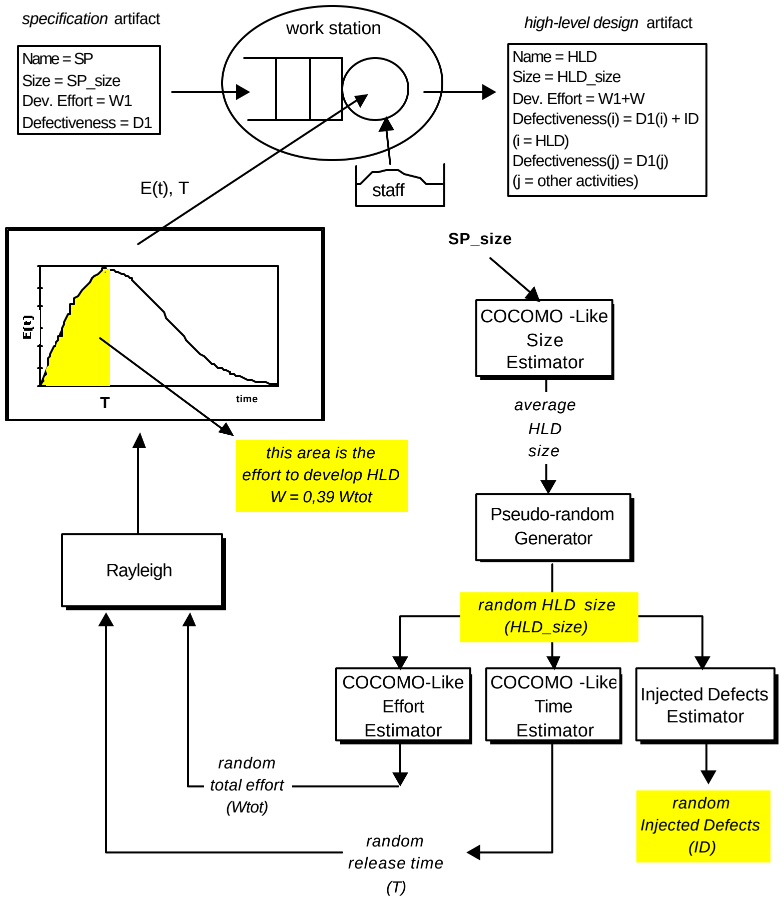

specification artifact
Name = SP
Size = SP_size
Dev. Effort = W1
Defectiveness = D1
work station
staff
high-level design artifact
Name = HLD
Size = HLD_size
Dev. Effort = W1+W
Defectiveness(i) = D1(i) + ID
(i = HLD)
Defectiveness(j) = D1(j)
(j = other activities)
E(t), T
E(t)
T
time
this area is the effort to develop HLD W = 0,39 Wtot
Rayleigh
SP_size
COCOMO -Like Size Estimator
average HLD size
Pseudo-random Generator
random HLD size (HLD_size)
COCOMO-Like Effort Estimator
COCOMO -Like Time Estimator
Injected Defects Estimator
random total effort (Wtot)
random release time (T)
random Injected Defects (ID)

Figure 4 - Lower abstraction level of the "work station" in the HLD activity

The amount of defects injected into the *high-level design* (injected defects, ID) is obtained through the injected defect estimator block, by multiplying the random *high- level design* size times the expected defect density (defects per unit of size, DD).

$$ID = HLD\_size \times DD$$

Defect density is a parameter used in DCM to summarise the effects of various factors (personnel skill, team structure, supporting tools, programming language, product type, etc) on the defectiveness of a given development activity. DCM, however, can easily accept more elaborate defect injection models [7].

The derived ID is then summed to D1 (*specification* defectiveness) to obtain the *high-level design* defectiveness.

More details on the analytical derivations of the functions used to model this station (and all the other stations in Figures 2 and 3) are in [4, 5, 6].

4. Applying the Model

The complexity of the resulting model has forced us to evaluate it by simulation.

The model has been applied to reproduce two possible software development scenarios. In the first scenario a stable set of requirements is assumed, in particular a *requirements* artifact of 1500 Function Points is given in input to the process model. In the second scenario, a certain amount of instability is assumed: *requirements increments* and *requirements changes* are regularly fed into the process, to reproduce a continuous requirements increment and change activity. Over the development time, the requirements growth from the initial amount of 1500FP to an amount of 1500FP+20%, while the 15% of the initial requirements is changed.

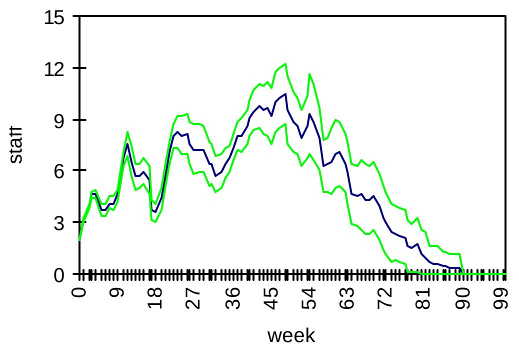
Simulation results for the first scenario are summarized in Table 1, columns 2 and 3, and in Figures 5 and 6.

Attribute	Results	Confidence (95%)	SEL data
Final Size	116 KLOC	+/- 20 KLOC	116 KLOC
Effort (W)	500 PW	+/- 60 PW	600 PW
Delivery Time (T)	78 W	+/- 3 W	63 W
Productivity (P)	5.8 LOC/P-hour	+/- 1.8 LOC/P-hour	5,3 LOC/P-hour
Rework Percentage (RWK)	17%	+/- 3%	/
Defect Density (DFD)	0.9 defects/KLOC	+/- 0.3 defects/KLOC	/
Average Staff	6.5 P	+/- 1 P	9.5 P

Table 1 – Simulation results in case of Stable Requirements

Results in Table 1 give the predicted final size of the product, the related effort, delivery time, productivity, rework percentage, defects density, and staff, with the corresponding confidence intervals. Figures 5 and 6 give the dynamics of the personnel during the development.

Figure 5 – Project Staffing Profile (stable requirements)



Verification of simulator validity is done by obtaining a good level of confidence in the representativeness of the simulation model against real-life situations. This is done in two ways: 1) demonstrating the model capability of reproducing empirically-known facts (in particular of reproducing the effects of requirements instability on process quality attributes), and 2) performing a quantitative and qualitative comparison against SEL data, i.e. models and actual projects.

To compare against SEL data, Table 1, in column 4, reports data obtained by using SEL estimation models, which show closeness to simulation results in column 2 plus/minus the confidence intervals in column 3. Closeness to SEL data is reinforced by Figures 5 and 6. The average staff profile for the entire project, shown in Figure 2 with its interval of confidence (95%), bears a strong resemblance with the trapezoidal shape expected for a SEL project. In addition, also the staff profile for a single activity (Figure 6 details the staff profile for the implementation activity) shows a close similarity with a Rayleigh curve (dotted line), as expected for a SEL project.

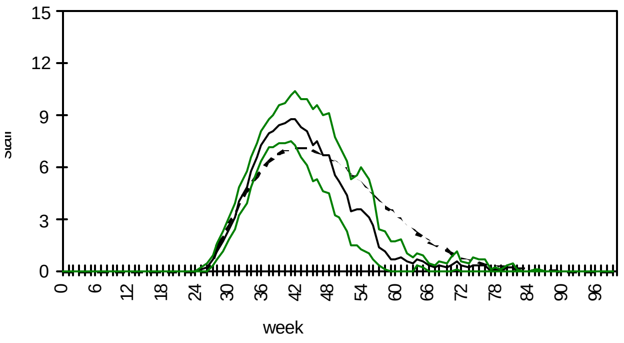


Figure 6 – Implementation Activity Staff Profile (stable requirements)

The model capability of reproducing empirically-known facts is demonstrated by the use of the simulator in case of unstable requirements, where the simulator confirms (see Figure 4) the empirical expectation that due to requirements instability a substantial increase of effort (W) and delivery time (T) is introduced, and provides a quantitative prediction of such an increment (38% and 60%, respectively). It also confirms the expected decrease of the process productivity (P), the increase of the rework percentage (RWK), and the deterioration of the final product quality (DFD). In fact, the rework percentage has more than doubled (a 150% increase), the productivity has clearly dropped (a 21% decrease), and the defect density of the final product has increased (a 66% increase).

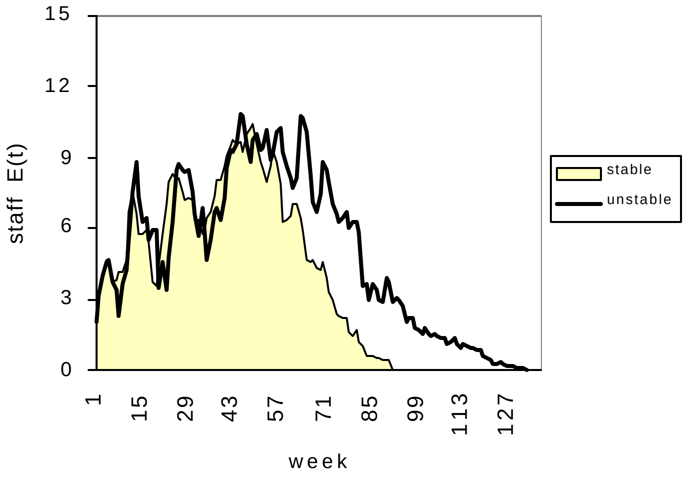


Figure 7 – Effects of requirements instability on the project staffing profile

The effects of requirements instability over the staffing profile illustrated in Figure 7 also gives an interesting comparison with SEL real project behavior. The profiles are similar in the two scenarios at the beginning of the project and diverge when the effects of the new and changed requirements increase. This behavior is qualitatively very similar to the staff profile measured by SEL for a project (Figure 8) with a final product size slightly larger than the simulated one (160 KLOC rather than 125 KLOC) and characterized by a high requirements instability

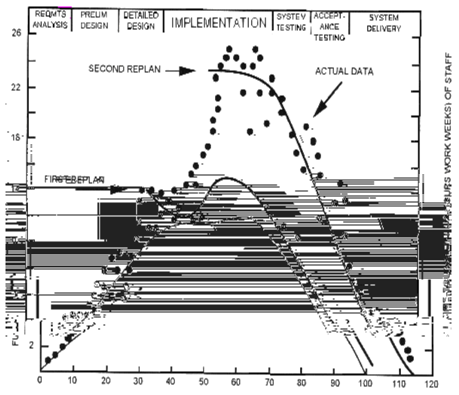


Figure 8 – SEL staffing profile for a project with high requirements instability

5. Conclusions

In conclusions, simulation results demonstrate the capability of the described model of reproducing empirically-known facts, and thus of being adopted as tool to test process assumptions. In particular, it can provide both qualitative and quantitative suggestions about how to tune the software process to improve its quality, i.e. to shorten delivery time, increase productivity, and reduce effort, rework and defects density.

Plan for future work includes the application of the suggested modelling method to less conventional process paradigms, such as the spiral paradigm and the concurrent engineering paradigm.

6. References

[1] Basili, V.R., Caldiera, G., Rombach, H.D., “The Experience Factory”, Encyclopædia of Software Engineering, Wiley&Sons Inc., 1994.

[2] Bohem B.W. Software Engineering Economics. Prentice-Hall, N.J., 1981.

[3] Calavaro, G.F., Basili V.R., Iazeolla G. "Simulation Modeling of Software Development Process", Proceedings of the 7th European Simulation Symposium, Erlangen-Nuremberg, GE, October 1995.

[4] Donzelli, P. and G. Iazeolla, “A Dynamic Simulator of Software Processes to Test Process Assumptions”, Journal of Systems and Software (to appear 1st quarter 2001).

[5] Donzelli, P. and G. Iazeolla, “A software Process Simulator for Software Product and Process Improvement”, Proceedings of the Product Focused Software Process Improvement Conference (Profes’99), Oulu, Finland, June 22-24, 1999.

[6] Donzelli, P. and G. Iazeolla, “Hybrid Simulation Modelling of the Software Process”, Software Process Simulation Modeling Workshop (ProSim 2000), 12-14 July 2000, London, UK.

[7] Kan, S. H. Metrics and Models in Software Quality Engineering. Addison-Wesley Publishing Company, MA, 1994.

[8] Martin, R.H., D. Raffo. "A model of the Software Development Process Using both Continuous and Discrete Models", International Journal of Software Process Improvement and Practice (to appear).

[9] Putnam, L.H. and W. Meyer. Measures for Excellence: Reliable Software on Time within Budget. Prentice-Hall, N.J., 1992.

[10] Rus, I., and J. Collefello, “Assessing the Impact of Defect Reduction Practices on Quality, Cost, and Schedule”, Software Process Simulation Modeling Workshop (ProSim 2000), 12-14 July 2000, London, UK.

[11] SEL-81-305, "Recommended Approach to Software Development", Revision 3. Software Engineering Laboratory Series, NASA-GSFC, Greenbelt, MD, 1992.

[12] SEL-84-101, "Manager's Handbook for Software Development", Revision 1. Software Engineering Laboratory Series, NASA-GSFC, Greenbelt, MD, 1990.

[13] SEL-94-002, "Software Measurement Guidebook", Software Engineering Laboratory Series, NASA-GSFC, Greenbelt, MD, 1994.

[14] SEL-95-105, "Software Process Improvement Guidebook", revision 1, Software Engineering Laboratory Series, NASA-GSFC, Greenbelt, MD, 1996.

Author Biographies

PAOLO DONZELLI is an Advisor with the Department of Informatics, Telecommunications and Statistics of the Office of the Prime Minister, Italy. Formerly, he served as an engineering officer with the Italian Air Force, and then he was with Cranfield University (UK), as senior research fellow. Dr Donzelli received a doctoral Laurea degree from the University of Naples “Federico II” (Italy), a MSc degree from the Cranfield University (UK), and a PhD degree from the University of Rome “TorVergata” (Italy). His research interests include software process improvement, requirements engineering and business process modeling.

GIUSEPPE IAZEOLLA is full professor of Computer Science, Software Engineering Chair, Faculty of Engineering, University of Rome “TorVergata” (Italy). His research is in the areas of software engineering and information system engineering, in relation to system performance and dependability modeling and validation.